White Paper

Introducing the Intel® Mobile Platform Software Development Kit (Intel® Mobile Platform SDK)

February 2005

Contents

Introduction 2

Challenges to Developing Mobile Applications 2

Overview of the Intel® Mobile Platform SDK 3

- Intel® Mobile Platform SDK Contents 3

Application Example 4

- Approach to Solving the Problem 4
- Solution to the Problem 4

Features Provided in the Intel® Mobile Platform SDK 6

Handling Network Disconnections 7

Summary 8

Introduction

Internet-connected mobile devices are rapidly becoming an integral part of the IT infrastructure of any thriving business. Users of notebook PCs, tablets, person digital assistants (PDAs) and cellular smart phones need their business applications to function on the road as well as in their offices. As a consequence, developers face the daunting task of extending existing applications across multiple platforms to handle the needs of users in this mobile computing environment.

The Intel® Mobile Platform Software Development Kit simplifies the task of adapting an application for use in a mobile environment. It provides a set of libraries and runtime components, along with a programming interface that is common across supported platforms and runtimes, to enable you to quickly and efficiently deliver applications with mobility features while maximizing code reuse.

Challenges to Developing Mobile Applications

Users of mobile devices expect applications to:

- **Allow working offline** – So users can access locally stored data and continue working when disconnected from the network and then resynchronize data between their device and the server when they reconnect.
- **Manage connectivity transparently** – So that users can focus on their core tasks rather than managing disconnections and reconnections to one or more networks.
- **Effectively balance power and performance** – So that users can get the most from their available battery life.
- **Work across multiple platforms** – So users can access an application on the device they choose while taking advantage of the capabilities of that device.

Two of the most significant challenges developers face when developing new mobile applications or extending existing applications to include mobile functionality are:

- Creating applications that are aware of platform context and resources and can efficiently take advantage of mobility features.
- Developing cross-platform, cross-runtime solutions to allow applications to be deployed across multiple clients.

When an application is capable of being aware of platform context, you can add features that enable the application to respond to environmental changes based on user preferences. For example, a user may select an option that specifies that the behavior of an application, such as a virus scanner, be optimized for performance when the laptop on which it is running is connected to external power and optimized for power when the laptop is running on batteries. When the laptop is disconnected from a power source, the virus scanner pauses a system-wide scan executing in the background to conserve battery power. It then resumes the scan when the power source is reconnected.

Mobilizing an application from scratch can be a daunting task requiring extensive coding and testing. Ideally, new mobile features should be developed in such a way that one set of code can be used across all supported platforms. A common programming interface with consistent object models, naming conventions and parameters across the programming languages being used can minimize code differences across platforms. Such an interface allows applications to be deployed natively on different platforms without extensive recoding.

Efficiency is increased and recoding reduced when you can work in your preferred runtime environment, independent of the platform for which the application is being developed or the language in which the application is written.

Overview of the Intel® Mobile Platform SDK

The Intel® Mobile Platform SDK provides you with tools to address all these challenges. Of particular importance are features of the SDK that support developing mobile applications that can run on multiple platforms and in various runtime environments, regardless of the runtime environment in which you choose to do your development.

Another important feature is the ability to make applications aware of and adapt to platform context and resources, so they can efficiently take advantage of all the mobility features of the platform. Runtime components included in the SDK monitor and provide automatic notification of context changes, allowing the application to respond to these changes programmatically in real time. This feature enables ISVs and users to specify and control the way a mobile platform reacts to changes in context.

The Intel Mobile Platform SDK includes libraries of information model components, redistributable runtime components, a *Programmer's Guide* describing the information model components, and several code samples and testing utilities. The Intel Mobile Platform SDK is supported on the Microsoft Windows* XP Professional and Microsoft Windows Mobile Edition 2003 platforms. Three versions of the programming interface are supported in the SDK: a Native C++ interface, a Sun Microsystems Java interface, and a Microsoft .NET* interface.

The common programming interface and model provided in the Intel Mobile Platform SDK is consistent across all the supported languages and platforms. This programming model includes:

- **Classes, instances, properties and methods** – Eight classes represent five devices and three system capabilities. The five devices modeled are a battery, a platform, a processor, a network adapter and a link protocol. The system capabilities include power, connectivity and bandwidth. Instances of these classes are described using properties and methods.
- **Events and threshold notifications** – Runtime components monitor system events and inform applications when an event occurs. For example, an application can "register" to be informed when the system goes on battery power. When this event occurs, the runtime component executes a callback routine to notify the application, which can than respond in real time. For example, the application may be programmed to go into a power-optimized mode to conserve power when the system goes on battery power. Event notification may be tied to a threshold, such as when the battery becomes discharged to a 50% level.
- **Collections and enumeration** – An ability to discover all the devices on the system allows a collection of multiple instances of a device on the system to be enumerated.

Intel Mobile Platform SDK Contents

The installed Intel Mobile Platform SDK contains the following directories and files:

- ***Docs* directory** – Contains the *Programmer's Guide*, a release notes, a directory containing the Sun Microsystems Javadoc* tool and legal documents.
- ***Development* directory** – Contains Native C++, Java and .NET libraries.
- ***Tools* directory** – Contains several tools to use with the SDK.
  - ***Information Viewer*** – A sample application that demonstrates features of the SDK such as:
    - Enumerating the devices on a system and viewing their properties.
    - Registering to be notified of an event and viewing the notification reported when the event occurs.

    You can use the Information Viewer as a validation tool to compare results to those returned by your implementation of Intel® Mobile Platform features in your application. Two viewers are provided, one written in .Net/C# and the other in Java.
  - ***Intel® Mobile Bandwidth Module*** – The Intel Mobile Bandwidth Module is a stand-alone Visual C# .NET application that is installed on the end-user client system with the SDK runtime components. It provides a simple graphical user interface to view the status of and manage network band width usage on an individual system. Online help is accessible from the interface.
- ***bin* directory** – Contains the executables that are installed on end-user client systems.
  - ***IMPPolicyService.exe*** – A runtime component that monitors and enforces polices. It is installed as a Windows service and is launched automatically when the system is booted.
  - ***IntelMobile_Server.exe*** – A runtime component that sends notifications of events to processes that have registered for them. It is launched automatically when an application that uses an SDK feature accesses the feature.

Application Example

An application that hasn't been designed for use in a mobile environment may exhibit behaviors frustrating for a user. For example, it may use excessive power when the system is running on batteries, decreasing the time the system can run on battery power between charges. The application example below shows how you can use features of the Intel Mobile Platform SDK to optimize the use of power by an application by pausing non-critical processes when the system is running on battery power.

Problem Statement: An application is unable to optimize its use of system power. You would like to add capability to the application to turn off processing of non-critical functions when the system is on battery power and to resume processing these functions when the system is connected to external AC power. For efficiency, you want to add this functionality without polling the operating system for the power connection status.

Approach to Solving the Problem

The Power capability provided in the Intel Mobile Platform SDK accesses information about the system power. The following features of the Power capability will be used in the solution to this problem:

Source property – The values the Source property are *External* and *Internal*.

Events – The two Power events that will be monitored are *InternalPower* and *ExternalPower*.

To solve this problem, you will incorporate the following functionality into your application:

1. At application startup, check the system connection status to determine if the application should initially be put in the low power mode or in the high performance mode.
2. Register for events that indicate the system power source has changed.
3. Respond appropriately when notification of an event is received.
   a. For an *InternalPower* event, pause all optional, non-critical processes, such as background virus checking or automatic word processor spell checking.
   b. For an *ExternalPower* event, resume processing of optional, non-critical processes that have been paused.
4. At application shutdown, unregister events and free allocated memory.

The next section will describe in detail how components and functions of the SDK are used to implement this solution.

Solution to the Problem

The procedure below provides details about how you can incorporate code into an application to optimize its use of system power. A version of this code that can be compiled is provided in an MSVC.NET 2003 project, so that you can download it and try it out. The MSVC.NET 2003 project is available at **www.intel.com/software/products/mobileplatform/**.

Steps include:

1. In the project that will use the Intel Mobile Platform SDK, include the appropriate header file and add the .lib file to the link stage.
   a. Add the header files directory to the include path for the project. By default this is: "C:\Program Files\Intel\Intel(R) Mobile Platform SDK\Development\C++".
   b. At the top of source files add:
   `#include "inc\IntelMobileCPP.h"`
   c. Add the lib files directory to the project Lib path. By default this is: "C:\Program Files\Intel\Intel(R) Mobile Platform SDK\Development\C++".
   d. Add the following to the linker input:
   `IntelMobileCPP.lib`
2. Add an initialization routine to run at application start-up that includes the following functions:
   a. Create a pointer to the PowerInstance object and initialize it. The PowerInstance object will be used throughout the life of the process to obtain information and receive notification of events related to power.

```
PowerInstance* pMyPowerInstance;

CapabilityClass MyClass;
pMyPowerInstance = reinterpret_cast<PowerInstance*>
         ( MyClass.GetInstance(L"Power") );
```

b. Get values describing the current status of the system power, such the power source currently being used (battery or external AC) and the percent of remaining system battery power. The IS NULL method is used in the code below to ensure that a particular feature is supported on the running platform.

```
unsigned int PowerInfo::GetPowerLevel()
{
  try
  {
    if (pMyPowerInstance != NULL)
    {
      if (!pMyPowerInstance->
         LifePercentRemaining.IsNull())
      {
        unsigned __int32 value = pMyPowerInstance->
           LifePercentRemaining.GetValue();
        return value;
      }
    }
  }
  catch (IntelMobileException& ex)
  {
    //Handle Exception here.
    ASSERT(0);
  }

  return 0;
}

CString PowerInfo::GetPowerSource()
{
  CString rv = _T("Unknown");

  try
  {
    if (pMyPowerInstance != NULL)
    {
      if (!pMyPowerInstance->Source.IsNull())
      {
        switch ( pMyPowerInstance->
           Source.GetValue() )
        {
          case Internal:
            rv = _T("Internal");
            break;
          case External:
            rv = _T("External");
            break;
        }
      }
    }
  }
  catch (IntelMobileException& ex)
  {
    //Handle Exception here.
    ASSERT(0);
  }

  return rv;
}
```

c. Define an Observer class called PowerObserver. An instance of this class will be registered to handle events. (See step 3 on the following page for how this class is implemented.)

```
#include "inc\Base\base_Event.h"
#include "inc\Base\base_Observer.h"
using namespace Intel::Mobile::Base;

class PowerObserver : public Observer
{
  public:
    void Notify( const Event& event);
    void SetDialogPtr(CDialog *DialogPtr);
  private:
    CDialog *MyUIDialog;
      //Used to update the user interface
};
```

d. Register an instance of the PowerObserver class called MyPowerObserver to be informed of power related events. In the code example below, MyPowerObserver is being registered for the ExternalPower, InternalPower and StatusChange events.

```
void PowerInfo::RegisterObserver()
{
  pMyPowerInstance->
     ExternalPower.AddObserver(MyPowerObserver);
  pMyPowerInstance->
     InternalPower.AddObserver(MyPowerObserver);
  pMyPowerInstance->
     StatusChange.AddObserver(MyPowerObserver);
}
```

3. Add a runtime event handler to receive notification of power-related changes to the system. In the code example below, three events are handled: a change to AC power (external), a change to battery power (internal), and a change in battery level. In this simplified example, the response is to post WM_PAINT, causing the main dialog box to update.

A more typical event handler for power events would respond to an event by changing the behavior of the application to optimize power use, such as turning off background processing when the system goes to battery power and turning it back on when AC power is reconnected.

The notify method runs on its own separate thread. Therefore, anytime the notify function is used to update shared data, critical sections must be implemented in the code to protect the shared data. Critical sections were not used in the simple code example below, but will be required for most applications.

```
#include "PowerObserver.h"
#include "inc\IntelMobileCPP.h"

using namespace Intel::Mobile::Base;
using namespace Intel::Mobile::Capability;

void PowerObserver::Notify(const Event& event)
{
  try
  {
    switch (event.GetType())
    {
      case Event::eExternalPower:
        OutputDebugString(_T("ExternalPower
           Event received.\n"));
        MyUIDialog->PostMessage(WM_PAINT);
           //Tell the main dialog to repaint.
        break;

      case Event::eInternalPower:
       OutputDebugString(_T("InternalPower
          Event received.\n"));
       MyUIDialog->PostMessage(WM_PAINT);
          //Tell the main dialog to repaint.
        break;

      case Event::eStatusChange:
        OutputDebugString(_T("StatusChange
           Event received.\n"));
        MyUIDialog->PostMessage(WM_PAINT);
           //Tell the main dialog to repaint.
        break;
    }
  }
  catch (IntelMobileException& ex)
  {
    OutputDebugString(_T("IntelMobileException
       exception caught!!!\n"));
  }
}
```

4. Add a cleanup routine to run at shutdown that unregisters observers for power-related events and removes them.

```
void PowerInfo::UnregisterObserver()
{
  pMyPowerInstance->
     ExternalPower.RemoveObserver(MyPowerObserver);
  pMyPowerInstance->
     InternalPower.RemoveObserver(MyPowerObserver);
  pMyPowerInstance->
     StatusChange.RemoveObserver(MyPowerObserver);
}
```

Features Provided in the Intel Mobile Platform SDK

The example above introduced a few of the devices and capabilities in the Intel Mobile Platform SDK and showed how they were used to enable an application to modify its behavior to optimize its use of system power. The SDK provides a set of devices and capabilities that can be used to solve a range of problems faced by mobile users.

The SDK supports the following devices:

- **NetworkAdapter** – Used to describe the network adapters currently available in the system, such as IEEE 802.3* (Ethernet) or WirelessLAN.
- **LinkProtocol** – Used to describe the link protocols running on the system, such as IEEE 802.11a*, and to monitor the state of network connections.
- **Battery** – Used to access information about a battery in the system, such as percent charge, life remaining and manufacturer's serial number.
- **Processor** – Used to access information about the system microprocessor, such as the manufacturer, the current processor frequency, and whether Intel® Streaming SIMD 2 Extensions (SSE2) or Intel® Streaming SIMD 3 Extensions (SSE3) are supported.
- **Platform** – Used to access information about the system platform, such as when the system enters suspend mode or hibernate mode or shutdown.

Capabilities provide aggregate information about all the devices on a system. For example, if a system has two batteries, the Power capability can be used to determine the average charge remaining in the two batteries. If one battery is at 90% charge and the other at 100%, the Power capability will report the aggregate charge as 95%.

A complete list of the capabilities supported by the SDK includes:

- **Power** – Used to access information about system power, such as the system power source (battery or external AC) or the aggregated system battery charge.
- **Connectivity** – Used to access system connectivity information, such as whether the system has at least one valid network interface connected to a network or whether a remote host is reachable.
- **Bandwidth** – Used to monitor and control the network bandwidth or transfer rate.

The Intel Mobile Platform SDK is supported on the Microsoft Windows* XP Professional and Microsoft Windows Mobile Edition 2003 platforms. It provides programming interfaces that are supported on Native C++, Java and .NET.

Handling Network Disconnections

Problem Statement: An application that hasn't been designed for mobility shows strange error messages and other abnormal behaviors when the client is disconnected from the server. The application must be restarted after the client is reconnected. You would like to add capability to the application to enable it to handle network disconnections and reconnections seamlessly.

Intel® Mobile Platform SDK components used to solve this problem:

ConnectivityInstance object

Connectivity events: *Connect*, *Disconnect*, *IpAddressChange*

The steps to solving this problem:

1. Get a pointer to a ConnectivityInstance object and check the current state of the connection to the network using the Connected property. If the network is available, run the application in a "connected" mode. If the network is not available, run the application in an "off-line" mode.
2. Register for the *Disconnect* event. The application registers an observer for the event so that it will be notified of a change in the state of the network connection. The Connectivity capability is used to monitor the state of network connections. It triggers a *Connect* event or *Disconnect* event when the state of the network connection changes. When Dynamic Host Configuration Protocol (DHCP) is being used and a system reconnect occurs, the *IpAddressChange* event is triggered when an IP address has been assigned. In some cases, this event is a better indication that the network connection is available for use.
3. Add features to your application to let the user know when the network connection state changes. For example, the application could display a message such as "The network has been disconnected. The data you are using has been saved locally". Or you could incorporate a status bar icon into your application that shows when the application is operating offline.
4. Modify your application to respond appropriately to changes in the network connection state. For example, in response to a *Disconnect* event, data could be cached locally and any changes made by the user saved locally. When the network connection is reestablished as indicated by a *Connect* event, the application could then push local data back up to the server and synchronize it.
5. Add a shutdown procedure to the application to unregister for the observer object for the *Connect*, *Disconnect* and *IpAddressChange* events when the application shuts down.

Summary

As users increasingly turn to mobile platforms, software developers face rising user dissatisfaction with applications that were not designed to work in a mobile environment. Adding mobile features to your applications will provide users with consistent application behavior optimized for a mobile environment.

The Intel Mobile Platform SDK helps you add mobile features to your applications while reducing development time and maximizing code reuse.

The Intel Mobile Platform SDK provides:

- Multiplatform/multiruntime support
- A common, consistent programming interface that provides access to key capabilities needed by mobile applications
- The ability to easily provide context awareness to applications

---

To evaluate the Intel Mobile Platform SDK, complete the form provided at **www.intel.com/software/products/mobileplatform/**. We will contact you shortly to arrange for an evaluation copy of the SDK to be made available to you.
